COP 4516: Competitive Programming and Problem Solving

Giri Narasimhan & Kip Irvine

Phone: x3748 & x1528

{giri,irvinek}@cs.fiu.edu

Evaluation

• Exam/Competition	50%
• Solving Problems	40%
• Attendance	5%
• Class Participation	5%

History of Algorithms

The great thinkers of our field:

- **Euclid, 300 BC**
- **Bhaskara, 6th century**
- **Al Khwarizmi, 9th century**
- **Fibonacci, 13th century**
- **Babbage, 19th century**
- **Turing, 20th century**
- **von Neumann, Knuth, Karp, Tarjan, …**

Al Khwarizmi's algorithm

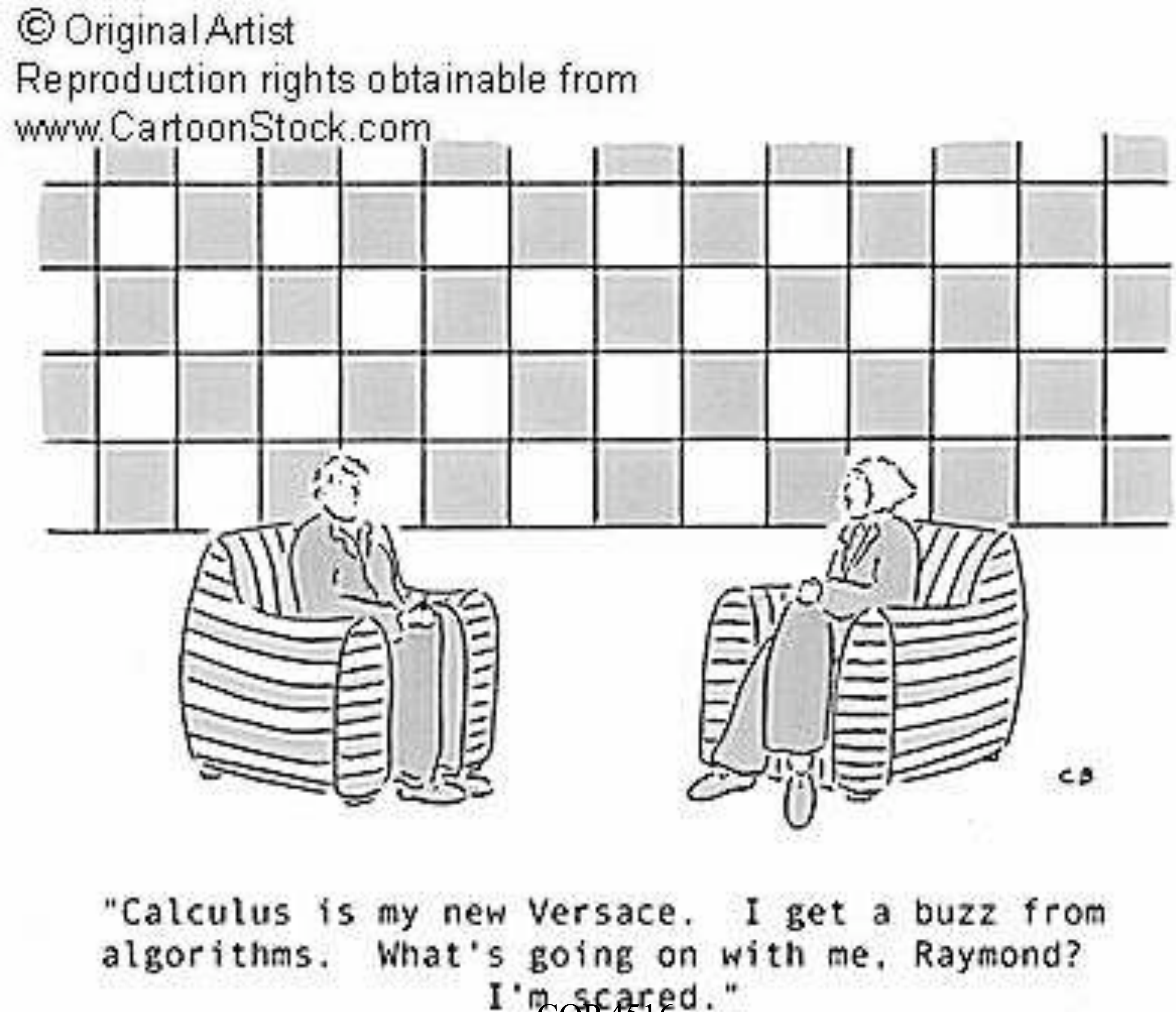
- 43 X 17
  - 43 17
  - 21 34
  - 10 68 (ignore)
  - 5 136
  - 2 272 (ignore)
  - 1 544

731

Euclid's Algorithm

- GCD(12,8) = 4; GCD(49,35) = 7;
- GCD(210,588) = ??
- GCD(a,b) = ??
- **Observation**: [a and b are integers and a ≥ b]
  - **GCD(a,b) = GCD(a-b,b)**
- **Euclid's Rule**: [a and b are integers and a ≥ b]
  - **GCD(a,b) = GCD(a mod b, b)**
- **Euclid's GCD Algorithm:**
  - **GCD(a,b)**
    If (b = 0) then return a;
    return GCD(a mod b, b)

If you like Algorithms, nothing to worry about!



Search

- You are asked to guess a number X that is known to be an integer lying in the range A through B. How many guesses do you need in the worst case?
  - Use **binary search**; Number of guesses = $\log_2(B-A)$
- You are asked to guess a positive integer X. How many guesses do you need in the worst case?
  - NOTE: No upper bound is known for the number.
  - Algorithm:
    - figure out B (by using **Doubling Search**)
    - perform binary search in the range B/2 through B.
  - Number of guesses = $\log_2 B + \log_2(B - B/2)$
  - Since X is between B/2 and B, we have: $\log_2(B/2) < \log_2 X$,
  - Number of guesses $< 2\log_2 X - 1$

Polynomial Evaluation

- Given a polynomial
  - $p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1} + a_n x^n$

  compute the value of the polynomial for a given value of x.
- How many additions and multiplications are needed?
  - Simple solution:
    - Number of additions = n
    - Number of multiplications = 1 + 2 + … + n = n(n+1)/2
  - Reusing previous computations: n additions and 2n multiplications!
  - Improved solution using **Horner's rule**:
    - $p(x) = a_0 + x(a_1 + x(a_2 + \ldots x(a_{n-1} + x\, a_n))\ldots))$
    - Number of additions = n
    - Number of multiplications = n

Sorting

- Input is a list of n items that can be compared.
- Output is an ordered list of those n items.
- Fundamental problem that has received a lot of attention over the years.
- Used in many applications.
- Scores of different algorithms exist.
- Task: To compare algorithms
  - On what bases?
    - Time
    - Space
    - Other

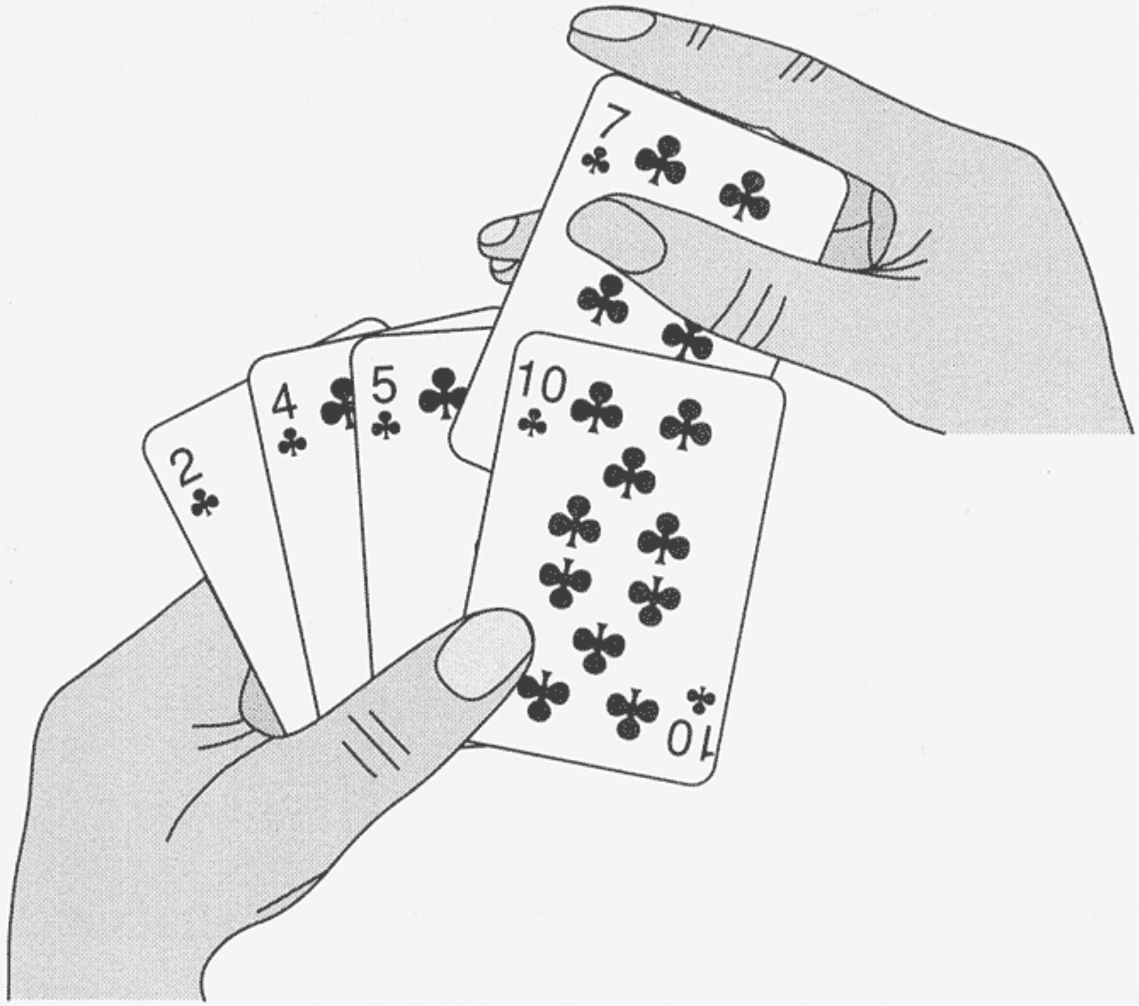


Figure 2.1 Sorting a hand of cards using insertion sort.

Sorting Algorithms

- Number of Comparisons
- Number of Data Movements
- Additional Space Requirements

Sorting Algorithms

- SelectionSort
- InsertionSort
- BubbleSort
- ShakerSort
- MergeSort
- HeapSort
- QuickSort
- Bucket & Radix Sort
- Counting Sort

SelectionSort

$\textsc{SelectionSort}(array\ A)$

$N \leftarrow length[A]$

for $p \leftarrow 1$ **to** N

  do Compute j, the index of the smallest item in $A[p..N]$

  Swap $A[p]$ and $A[j]$

SelectionSort

```
SelectionSort(array A)
N ← length[A]
for p ← 1 to N
      do ▷ Compute j
         j ← p
         for m ← p + 1 to N
             do if (A[m] < A[j])
                   then j ← m
         ▷ Swap A[p] and A[j]
         temp ← A[p]
         A[p] ← A[j]
         A[j] ← temp
```

SelectionSort

```
SelectionSort(array A)
N ← length[A]
for p ← 1 to N
      do ▷ Compute j
         j ← p
         for m ← p + 1 to N
             do if (A[m] < A[j])
                   then j ← m
         ▷ Swap A[p] and A[j]
         temp ← A[p]
         A[p] ← A[j]
         A[j] ← temp
```

$O(n^2)$ time

$O(1)$ space

Solving Recurrence Relations

Page 62, [CLR]

Recurrence; Cond	Solution
$T(n) = T(n-1) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-1) + O(n)$	$T(n) = O(n^2)$
$T(n) = T(n-c) + O(1)$	$T(n) = O(n)$
$T(n) = T(n-c) + O(n)$	$T(n) = O(n^2)$
$T(n) = 2T(n/2) + O(n)$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a = b$	$T(n) = O(n \log n)$
$T(n) = aT(n/b) + O(n)$; $a < b$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a - \epsilon})$	$T(n) = O(n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = O(n^{\log_b a})$	$T(n) = \Theta(n^{\log_b a} \log n)$
$T(n) = aT(n/b) + f(n)$; $f(n) = \Theta(f(n))$ $af(n/b) \leq cf(n)$	$T(n) = \Omega(n^{\log_b a} \log n)$

INSERTION-SORT(A)

```
for j ← 2 to length[A]
    do key ← A[j]
       ▷ Insert A[j] into the sorted sequence A[1 . . j − 1].
       i ← j − 1
       while i > 0 and A[i] > key
           do A[i + 1] ← A[i]
              i ← i − 1
       A[i + 1] ← key
```

Loop invariants and the correctness of insertion sort

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** *length*$[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
$\triangleright$ Insert $A[j]$ into the sorted sequence $A[1 \,..\, j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

O(n^2) time

O(1) space

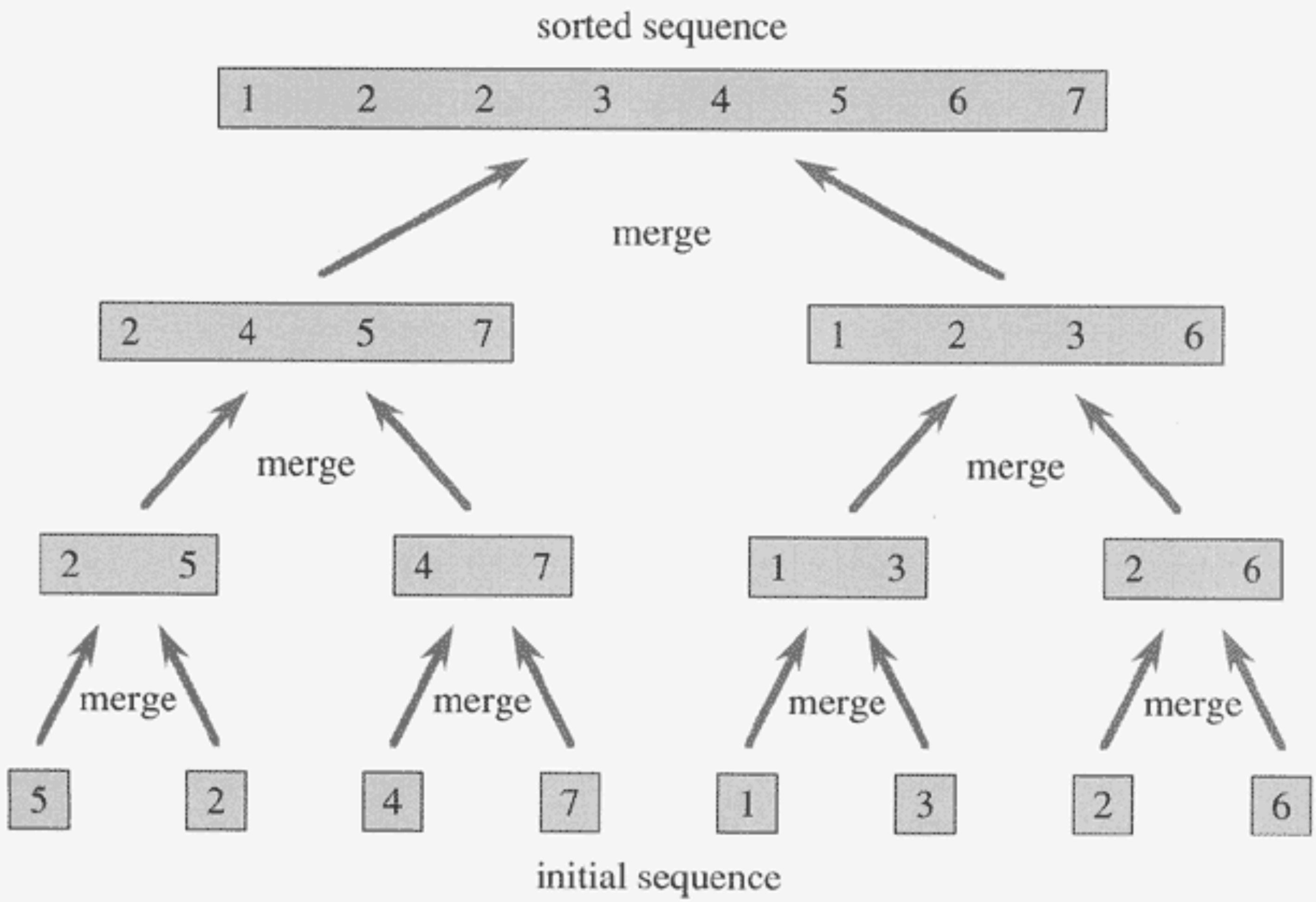
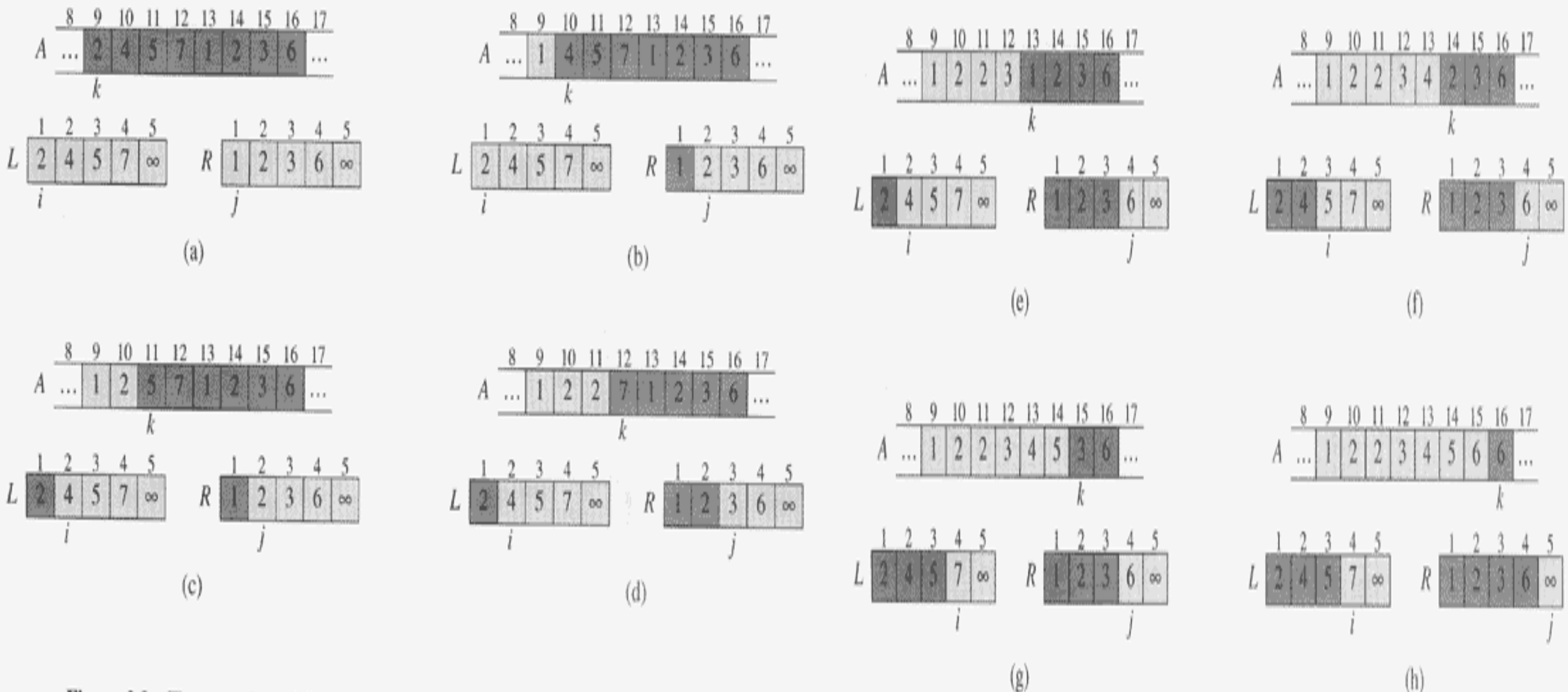


Figure 2.4 The operation of merge sort on the array $A = \langle 5, 2, 4, 7, 1, 3, 2, 6 \rangle$. The lengths of the sorted sequences being merged increase as the algorithm progresses from bottom to top.



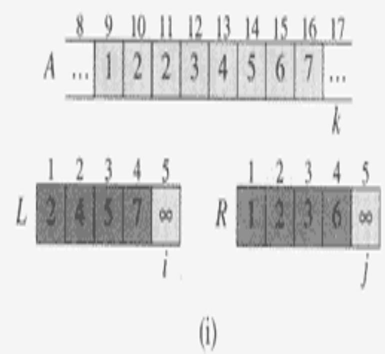


Figure 2.3 The operation of lines 10–17 in the call MERGE$(A, 9, 12, 16)$, when the subarray $A[9..16]$ contains the sequence $\langle 2, 4, 5, 7, 1, 2, 3, 6 \rangle$. After copying and inserting sentinels, the array L contains $\langle 2, 4, 5, 7, \infty \rangle$, and the array R contains $\langle 1, 2, 3, 6, \infty \rangle$. Lightly shaded positions in A contain their final values, and lightly shaded positions in L and R contain values that have yet to be copied back into A. Taken together, the lightly shaded positions always comprise the values originally in $A[9..16]$, along with the two sentinels. Heavily shaded positions in A contain values that will be copied over, and heavily shaded positions in L and R contain values that have already been copied back into A. **(a)–(h)** The arrays A, L, and R, and their respective indices k, i, and j prior to each iteration of the loop of lines 12–17. **(i)** The arrays and indices at termination. At this point, the subarray in $A[9..16]$ is sorted, and the two sentinels in L and R are the only two elements in these arrays that have not been copied into A.

```
MERGE(A, p, q, r)
n1 ← q − p + 1
n2 ← r − q
create arrays L[1 .. n1 + 1] and R[1 .. n2 + 1]
for i ← 1 to n1
     do L[i] ← A[p + i − 1]
for j ← 1 to n2
     do R[j] ← A[q + j]
L[n1 + 1] ← ∞
R[n2 + 1] ← ∞
i ← 1
j ← 1
for k ← p to r
     do if L[i] ≤ R[j]
           then A[k] ← L[i]
                i ← i + 1
           else A[k] ← R[j]
                j ← j + 1
```

Assumption: Array A is sorted from positions p to q and also from positions q+1 to r.

MERGE-SORT(A, p, r)

if $p < r$

  then $q \leftarrow \lfloor (p + r)/2 \rfloor$

    MERGE-SORT(A, p, q)

    MERGE-SORT$(A, q + 1, r)$

    MERGE(A, p, q, r)

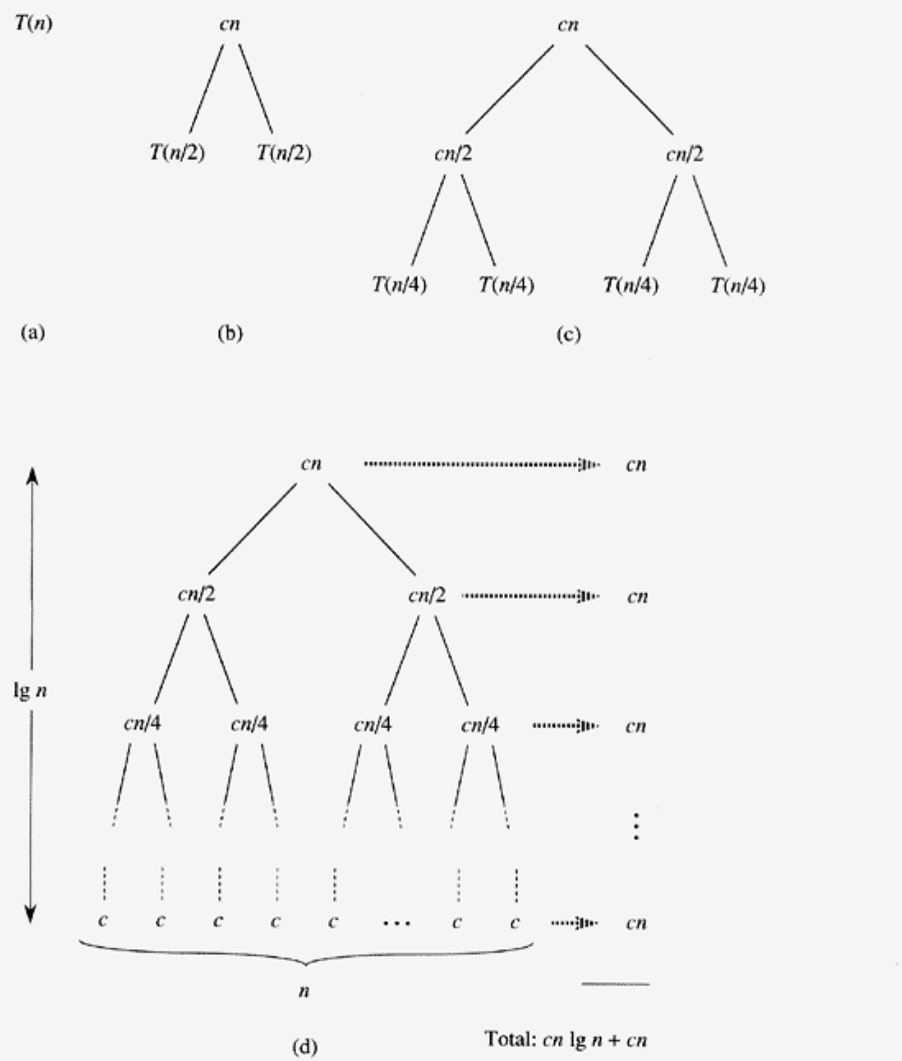


Figure 2.5 The construction of a recursion tree for the recurrence $T(n) = 2T(n/2) + cn$. Part **(a)** shows $T(n)$, which is progressively expanded in **(b)–(d)** to form the recursion tree. The fully expanded tree in part (d) has $\lg n + 1$ levels (i.e., it has height $\lg n$, as indicated), and each level contributes a total cost of cn. The total cost, therefore, is $cn \lg n + cn$, which is $\Theta(n \lg n)$.